

**NETWORK INTERFACE CARD USING PHYSICAL LAYER  
MICROCONTROLLER AND METHOD OF OPERATION**

**Inventor(s) :**

John E. Gavlik  
2540 Greencastle Court  
Oxnard  
Ventura County  
California 93035-2901  
United States Citizen

Ted Chang  
5355 Oak Park Lane, No. 170  
Oak Park  
Los Angeles County  
California 91377  
United States Citizen

Matthew J. Webb  
5323 Baza Avenue  
Woodland Hills  
Los Angeles County  
California 91364  
Citizen of the United Kingdom

**Assignee:**

National Semiconductor Corporation  
2900 Semiconductor Drive  
Santa Clara, California 58090

**CERTIFICATE OF EXPRESS MAIL**

I hereby certify that this correspondence, including the attachments listed, is being mailed in an envelope addressed to Commissioner of Patents and Trademarks, Washington, DC 20231, using the Express Mail Post Office to Addressee service of the United States Postal Service on the date shown below.

KATHY LONGENECKER  
Printed Name of Person Mailing  
Kathy Longenecker  
Signature of Person Mailing

EE 80453013345  
Express Mail Receipt No.  
11/15/00  
Date

William A. Munck  
John T. Mockler  
Novakov Davis & Munck P.C.  
900 Three Galleria Tower  
13155 Noel Road  
Dallas, Texas 75240  
(214) 922-9221

007433344500

NETWORK INTERFACE CARD USING PHYSICAL LAYER  
MICROCONTROLLER AND METHOD OF OPERATION

CROSS-REFERENCE TO RELATED APPLICATIONS

10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140 145 150 155 160 165 170 175 180 185 190 195 200 205 210 215 220 225 230 235 240 245 250 255 260 265 270 275 280 285 290 295 300 305 310 315 320 325 330 335 340 345 350 355 360 365 370 375 380 385 390 395 400 405 410 415 420 425 430 435 440 445 450 455 460 465 470 475 480 485 490 495 500 505 510 515 520 525 530 535 540 545 550 555 560 565 570 575 580 585 590 595 600 605 610 615 620 625 630 635 640 645 650 655 660 665 670 675 680 685 690 695 700 705 710 715 720 725 730 735 740 745 750 755 760 765 770 775 780 785 790 795 800 805 810 815 820 825 830 835 840 845 850 855 860 865 870 875 880 885 890 895 900 905 910 915 920 925 930 935 940 945 950 955 960 965 970 975 980 985 990 995

The present invention is related to that disclosed in United States Patent Application Serial No. [ATTY DOCKET NO. P04762], entitled "MULTITASKING MICROCONTROLLER FOR CONTROLLING THE PHYSICAL LAYER OF A NETWORK INTERFACE CARD AND METHOD OF OPERATION" and filed concurrently herewith. The above application is commonly assigned to the assignee of the present invention. The disclosure of this related patent application is hereby incorporated by reference into the present disclosure as if fully set forth herein.

TECHNICAL FIELD OF THE INVENTION

The present invention is generally directed to network interface cards and, more specifically, to a microcontroller architecture that controls the physical layer of a network interface card.

## BACKGROUND OF THE INVENTION

The demand for high-performance computers and communication devices requires that state-of-the-art networks and network interface devices operate at comparable high-performance levels.

5 The necessary high-performance is provided by network interface cards (NIC) that operate at ever increasing speeds. These network interface cards (NIC) are used in a wide variety of devices, including personal computers, switches, routers, hubs, bridges, and the like. Network interface cards operating at 10 Mbps (i.e., 10BaseT) over Category-3 (CAT3) wires and network cards operating at 100 Mbps (i.e., 100BaseT) over Category-5 (CAT5) are in common use in Ethernet local area network (LAN) environments. Additionally, network interface cards that operate at 1 Gbps (i.e., 1000BaseT) are now coming into use in Gigabit Ethernet LANs.

15 To achieve the desired high-speed performance, it is essential that the monitoring and control functions of network interface cards operate as real-time functions. Conventional network interface cards obtain real-time performance using hard-wired state machines to control and monitor the internal operations of the physical layer of the network interface cards. In effect, each individual function of the physical layer requires its own state

20

machine. This allows many functions to operate in parallel at very high speed.

However, the state machine approach has significant drawbacks. If a bug is found in a network card, if an existing function is to be modified, or if a new function is to be added, a new state machine must be designed or an existing state machine must be modified. Thus, a network interface card with hardwired state machine cannot be upgraded or patched and must be replaced. Some network card manufacturers attempt to overcome these drawbacks by using state machines that are at least partially implemented using PAL and PLA circuits. However, the degree to which a PAL or PLA circuit can be reprogrammed or upgraded is relatively limited. Moreover, many manufacturers use dedicated pins to communicate with and reprogram or upgrade the physical layer circuitry in their network interface cards. However, many network interfaces have only a limited number of I/O pins. Dedicating pins for reprogramming or upgrading purposes in these circumstances limits the versatility of a network interface card.

There is therefore a need in the art for improved network interface cards that may be easily upgraded or modified. In particular, there is a need for network interface card that may be upgraded or modified without using dedicated interface pins to

download software patches or reprogram state machines in the network cards. More particularly, there is a need for network interface cards in which the functions of the physical layer circuitry can be monitored and controlled in real time without  
5 using hard-wired state machines. There is a still further need for network interface cards in which the physical layer may be easily reprogrammed or upgraded without using PAL or PLA circuits to control state machines.

## SUMMARY OF THE INVENTION

5 To address the above-discussed deficiencies of the prior art, it is a primary object of the present invention to provide an apparatus for updating, correcting and enhancing the operation of a physical layer interface of a network interface card. According to an advantageous embodiment of the present invention, the apparatus comprises: 1) a read only memory (ROM) capable of storing an embedded control program; 2) a random access memory capable of storing a downloadable software control program; and 3) a microcontroller capable of controlling the physical layer interface, wherein the microcontroller in a first operating mode executes the embedded control program to thereby control the physical layer interface, and wherein the microcontroller in a second operating mode is capable of downloading the downloadable software control program from an external processing system and executing the software control program in place of the embedded control program to thereby control the physical layer interface.

15 According to one embodiment of the present invention, the ROM is an internal ROM in the microcontroller.

20 According to another embodiment of the present invention, the RAM is an internal RAM in the microcontroller.

According to still another embodiment of the present invention, the ROM is an external ROM coupled to the microcontroller.

According to yet another embodiment of the present invention,  
5 the RAM is an external RAM coupled to the microcontroller.

According to a further embodiment of the present invention, the microcontroller downloads the downloadable software control program from the external processing system via a media independent clock (MDC) signal line and a media independent input/output (MDIO) signal line.

According to a still further embodiment of the present invention, the microcontroller downloads the downloadable software control program via a medium access control (MAC) layer interface coupling the external processing system and the physical layer interface.

According to a yet further embodiment of the present invention, the microcontroller further comprises a plurality of control registers capable of controlling the first and second operating modes, wherein the microcontroller switches from the  
20 first operating mode to the second operating mode when the external processing system stores a jump address in one particular register in a first one of the plurality of control registers.

0971033363450  
The foregoing has outlined rather broadly the features and technical advantages of the present invention so that those skilled in the art may better understand the detailed description of the invention that follows. Additional features and advantages of the invention will be described hereinafter that form the subject of the claims of the invention. Those skilled in the art should appreciate that they may readily use the conception and the specific embodiment disclosed as a basis for modifying or designing other structures for carrying out the same purposes of the present invention. Those skilled in the art should also realize that such equivalent constructions do not depart from the spirit and scope of the invention in its broadest form.

Before undertaking the DETAILED DESCRIPTION OF THE INVENTION below, it may be advantageous to set forth definitions of certain words and phrases used throughout this patent document: the terms "include" and "comprise," as well as derivatives thereof, mean inclusion without limitation; the term "or," is inclusive, meaning and/or; the phrases "associated with" and "associated therewith," as well as derivatives thereof, may mean to include, be included within, interconnect with, contain, be contained within, connect to or with, couple to or with, be communicable with, cooperate with, interleave, juxtapose, be proximate to, be bound to or with, have,



have a property of, or the like; and the term "controller" means any device, system or part thereof that controls at least one operation, such a device may be implemented in hardware, firmware or software, or some combination of at least two of the same. It should be noted that the functionality associated with any particular controller may be centralized or distributed, whether locally or remotely. Definitions for certain words and phrases are provided throughout this patent document, those of ordinary skill in the art should understand that in many, if not most instances, such definitions apply to prior, as well as future uses of such defined words and phrases.

## BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the present invention, and the advantages thereof, reference is now made to the following descriptions taken in conjunction with the accompanying drawings, wherein like numbers designate like objects, and in which:

FIGURE 1 illustrates an exemplary processing system, namely a personal computer (PC), containing a network interface card (NIC) that incorporates a microcontroller in the physical layer hardware according to the principles of the present invention;

FIGURE 2 illustrates an exemplary network interface card (NIC) in greater detail according to one embodiment of the present invention;

FIGURE 3 illustrates an exemplary microcontroller in the physical layer controller of the network interface card according to one embodiment of the present invention;

FIGURE 4 is an exemplary memory map of the ROM and the RAM in the exemplary microcontroller according to the principles of the present invention;

FIGURE 5 is a flow diagram illustrating the operation of the exemplary network interface card according to one embodiment of the present invention;

FIGURE 6 is a flow diagram illustrating the hierarchy of multitasking routines in the control program that operates the exemplary network interface card according to one embodiment of the present invention; and

5        FIGURES 7-9 illustrate multitasking program flow in three exemplary subroutines according to one embodiment of the present invention.

## DETAILED DESCRIPTION OF THE INVENTION

FIGURES 1 through 9, discussed below, and the various embodiments used to describe the principles of the present invention in this patent document are by way of illustration only and should not be construed in any way to limit the scope of the invention. Those skilled in the art will understand that the principles of the present invention may be implemented in any suitably arranged network interface device.

FIGURE 1 illustrates an exemplary processing system, namely personal computer (PC) 100, containing a network interface card that incorporates a microcontroller in the physical layer hardware in accordance with the principles of the present invention. Personal computer 100 comprises removable (i.e., floppy) disk drive (FDD) 102 and hard disk drive (HDD) 103, monitor 104, keyboard 105, processor (CPU) 106, main memory 107, a pointing device, such as mouse 108, and network interface card (NIC) 140. Monitor 104, keyboard 105, and mouse 108 may be replaced by, or combined with, other input/output (I/O) devices. Main memory 107 may comprise a volatile storage device, such as a dynamic random access memory (RAM). Processor 106 may comprise an on-board two level cache system, including a Level 1 (L1) cache and a Level 2 (L2) cache.

Keyboard 105 and mouse 108 are coupled to PC 100 via input/output (I/O) interface (IF) 110. Monitor 104 is coupled to PC 100 via video/audio interface (IF) 112. The internal components of PC 100, including floppy disk drive 102, hard disk drive 103, processor 106, main memory 107, I/O interface 110 and video/audio interface 112, are coupled to and communicate across communication bus 115. Communication bus 115 may represent one or more buses in PC 100, including a Peripheral Component Interconnect (PCI) bus. Removable disk drive 102 is capable of reading and writing to removable floppy diskettes. Hard disk drive 105 provides fast access for storage and retrieval of application programs and data.

NIC 140 allows PC 100 to communicate with an external data network, such as a local area network (LAN) in an office. NIC 140 may operate at different speeds according to conditions on the external network. For instance, NIC 140 may operate in 10BaseT, 100BaseT, and 1000BaseT modes in which NIC 140 transfers data at 10 Mbps, 100 Mbps, and 1000 Mbps, respectively. In an advantageous embodiment, hard disk drive 103 stores network interface card (NIC) configuration file 103. As will be explained below in greater detail, NIC configuration file 103 comprises a software control program that may be downloaded into a random access memory (RAM) in the physical layer of NIC 140. A microcontroller according to the

principles of the present invention then executes the downloaded software control program in lieu of the embedded control program stored in read-only memory (ROM) that normally is executed by the microcontroller. In an advantageous embodiment of the present invention, whenever PC 100 is rebooted, software drivers executed by processor 106 retrieve and download the software control program in NIC configuration file 103 to the physical layer microcontroller of NIC 140.

FIGURE 2 illustrates exemplary network interface card (NIC) 140 in greater detail according to one embodiment of the present invention. Network interface card (NIC) 140 comprises 10-100-1000 Ethernet medium access control (MAC) layer controller 210 (hereafter, "MAC layer controller 210"), 10-100-1000 Ethernet physical layer controller 220 (hereafter, "physical layer controller 220"), magnetic (MAG) circuits 230, and connector 240, which may be, for example, an RJ45 connector. MAG circuits 230 comprises coupling transformer coils that transmit high bit-rate signals to, and receive high bit-rate signals from, for example, a Gigabit Ethernet LAN connected to RJ45 connector 240. Network interface card (NIC) 140 may also comprise optional external ROM 260 and optional external RAM 270. In the exemplary embodiment, ROM 260 and RAM 270 are both 16 kilobytes in size.

PC 100 communicates with and controls MAC layer controller 210 and physical layer controller 220 via PCI bus 250. The primary data interface between MAC layer controller 210 and physical layer controller 220 is the Media Independent Interface/Gigabit Media Independent Interface (MII/GMII). In an advantageous embodiment of the present invention, MAC layer controller 210 also communicates with physical layer controller 220 using the Management Data Clock (MDC) and Management Data Input/Output (MDIO) signal lines. The MDC and MDIO interface is a well-known, standardized interface used in most network interface cards to communicate between the MAC layer and the physical layer. In conventional network interface cards, MAC layer controller 210 uses the MDC/MDIO signal lines to access one or more of thirty-two (32) standardized registers (R0 through R31) in physical layer controller 220 for normal configuration and control functions, such as setting the bit rate (e.g., 10BaseT, 100BaseT, 1000BaseT), examining status registers, and the like.

In an exemplary embodiment of the present invention, MAC layer controller 210 may comprise a DP83820 chip from National Semiconductor Corporation and physical layer controller 220 may comprise a DP83861 chip from National Semiconductor Corporation. Those skilled in the art will understand, however, that equivalent

MAC layer and physical layer chips from other manufacturers may readily be adopted for use in accordance with the principles of the present invention.

According to the principles of the present invention, PC 100  
5 uses NIC driver software in hard disk drive 103 to download a new software control program from NIC configuration file 103 into physical layer controller 220 after PC 100 is booted up or reset. PC 100 uses MAC layer controller 210 and the MDC/MDIO signal lines to load the new software control program through selected ones of the R0-R31 registers and into random access memory (RAM) in physical layer controller 220. The new software control program is executed in place of the original embedded program in ROM in physical layer controller 220.

FIGURE 3 illustrates exemplary microcontroller 300 in physical layer controller 220 according to one embodiment of the present invention. Microcontroller 300 comprises microcontroller core logic 310, internal read-only memory (ROM) 320, internal random access memory (RAM) 330, management interface logic 340, and registers, peripheral logic 350, optional external ROM 260, and  
20 optional external RAM 270. Registers and peripheral logic 350 comprises RS-232 UART 352, computer operating properly (COP) timer 354, general purpose ports 356, and IEEE and expanded



registers 358. Microcontroller core logic 310 is coupled to ROM 320, RAM 330, ROM 260, RAM 270, and registers and peripheral logic 350 by address, data and control busses. Microcontroller core logic 310 also receives interrupt signals from management interface logic 340 and registers and peripheral logic 350.

In an exemplary embodiment of the present invention, microcontroller 300 may comprise a variation of a standard Motorola™ MC68HC11 microcontroller. Those skilled in the art will understand, however, that equivalent microcontrollers from other manufacturers may readily be adopted for use in accordance with the principles of the present invention. Furthermore, in the exemplary embodiment, ROM 320 and RAM 330 are each 16 kilobytes (16K) in size and are internal to microcontroller 300. However, this is by way of illustration only. In alternate embodiments, additional ROM 260 and RAM 270 may be external devices coupled to microcontroller 300 and the size of ROM 320, ROM 260, RAM 330, or RAM 270 also may be smaller or larger than 16 kilobytes.

In an exemplary embodiment, microcontroller core logic 310 comprises a high performance, synthesizable 8-bit CPU core. Microcontroller core logic 310 may implement, for example, the complete Motorola MC68HC11 instruction set and hardware architecture, including a sequencer, instruction decode unit,

arithmetic unit and registers, as well as other support logic. Microcontroller core logic 310 may include an interrupt priority resolution system. In an exemplary embodiment, microcontroller 300 is driven by a 41.667 MHz clock signal.

5           In normal operating modes, microcontroller 300 uses an internal embedded program (i.e., firmware) in ROM 320 to control 10Base-T, 100Base-T and 1000Base-T physical layer functions, as well as RS-232 and management communications with PC 100 and the external network. The ROM 320 firmware may perform the following major functions:

Power-On Initialization  
Start Up Configuration  
Main Loop Control  
Test Mode Control  
Loopback Control  
Auto Negotiation  
10/100/1000 Base PHY-Control  
1000 Base Link Monitor  
RS-232 Serial Communications  
20           Management Communications  
LED Illumination Control  
PATCH Routines

Normally, program variables, pointers, multitasking vectors and the stack reside in the 16Kbytes of RAM 320. According to the principles of the present invention, patches, upgrades and enhancements to the software control program may be downloaded as software (as opposed to firmware) that is stored in RAM 320 through management interface logic 340. The downloaded software code also permits the user to conduct extensive testing and debugging using the register interface of microcontroller 300.

RS-232 UART 352 forms a serial I/O (SIO) hardware interface between microcontroller 300 and PC 100. RS-232 UART 352 logic has two functional interfaces: a 2-wire link (RX and TX) to external host PC 100 and a 4 byte-wide data paths to microcontroller core logic 310. The TX and RX signals form a conventional RS-232 asynchronous communication link. Serial data can be transferred to and from PC 100 in full-duplex mode at one of four standard baud rates (115,200, 57,600, 38,400 and 19,200). Each serial data word is composed of a start bit, 8 data bits and one stop bit. Since this is a universally accepted asynchronous serial data format, it ensures that any terminal program resident on PC 100 can transmit and receive serial data to and from RS-232 UART 352 at the standard baud rates.

Computer operating properly (COP) timer 354 has two basic

functions: 1) to issue an interrupt if, and when, it is not properly serviced by firmware; and 2) to act as a general-purpose event timer for any firmware or software routine. At the core of COP timer 354 is a 26-bit free-running binary up counter that is incremented on each positive-going edge of the 41.67 MHz microcontroller clock. At a clock rate of 41.67 MHz, 26 bits are necessary due to the fact that microcontroller 300 may time events over 1.5 seconds.

The primary function of COP timer 354 is to indirectly detect software errors by timing out. Therefore, if the firmware and/or software are functioning correctly, COP timer 354 is periodically reset thus keeping it from timing out. Resetting COP timer 354 is accomplished by writing a Logic 1 to bit 0 of a COP control register. The Logic 1 write to this register is self clearing and COP timer 354 resumes counting from zero on the next positive-going edge of the 41.67 MHz microcontroller clock.

If COP timer 354 does time out, it is an indication that the firmware or software is no longer being executed in the intended manner. If a time-out occurs, COP timer 354 issues a non-maskable interrupt to microcontroller core logic 310 that resets the microcontroller firmware code back to a power-on reset condition. In addition to its primary watchdog function, COP timer 354 may

also be used as a general-purpose event timer. Since COP timer 354 is 26 bits in length and increments on each positive-going edge of the 41.67 MHz clock, microcontroller 300 can time events as long as 1.6 seconds.

5 IEEE and expanded registers 358 allow PC 100 to access the internal workings of microcontroller 300. In an exemplary embodiment, IEEE and expanded registers 358 are organized as 256 register files (RF0 through RF255) with each register file consisting of 64 bytes. Although all 256 register files may not be used, unused registers allow future functions to be added to the operation of microcontroller 300. Most individual registers exist in internal RAM with the exception of certain hardware-based registers normally located in RF0 through RF3.

Management interface logic 340 communicates with PCI bus 250 or PC 100 via the serial bit stream MDIO and the clock MDC with a specified frame structure and protocol as defined below:

Frame Unit	Format	Bits	Driver	Definition
Preamble	111...1	32	STA	A Preamble may be sent at the beginning of each transaction, consisting of 32 contiguous logic ones; this is optional
Start of Frame	01	2	STA	A '01' delimiter initiates the transaction
Operation Code	CC	2	STA	Read (10) or Write (01)
PHY Address	AAAAA	5	STA	A 5-bit PHY address with MSB first
Register Address	RRRRR	5	STA	A 5-bit register address with MSB first
Turn Around	NN	2	STA	A 2-bit turnaround time to avoid contention during Reads. Read (1Z) or Write (01)
Data	DDD...D	16	STA/ PHY	16-bits of data (MSB first) driven by the PHY on Read transactions or the STA on Write transactions
Idle	ZZZ...Z	>=0	STA	The idle condition is a high-impedance state. The PHY pulls-up the MDIO line to a logic one. No Idles are actually required

Using the above protocol, data can be read from or written into the registers of microcontroller 300 and RAM 330.

In conventional network interface cards, the MDC/MDIO interface is intended to access only thirty-two, 16-bit registers

in register file 0 (RF0) of IEEE and expanded registers 358. However, the present invention expands the use of this interface to incorporate reading and writing the entire 64K-memory space of microcontroller 300 via a direct memory access (DMA) hardware mechanism built into management interface logic 340. This expanded feature permits downloading new multi-tasking firmware into RAM 330 for the purpose of bug fixing, adding enhancements and internal diagnostics.

An expanded management interface according to the principles of the present invention can be viewed as having two modes of operation - base mode and expanded mode. Base mode management access can be described as a 16-bit read or write access to any register from register 0 (R0) to register 31 (R31) in the RF0 (base) register space. Expanded mode management accesses uses four (4) unused base mode registers as portals to the entire 64Kbyte register and memory space of microcontroller 300.

Expanded mode management access uses the following registers:

- |    |         |      |              |
|----|---------|------|--------------|
| 1) | RF0.R22 | 0x16 | Mode_Port    |
| 2) | RF0.R29 | 0x1D | Data_Port    |
| 3) | RF0.R30 | 0x1E | Addr_Port    |
| 4) | RF0.R31 | 0x1F | JMP_JSR_Port |

and operates in the following four modes:

Mode 1 (8-bit access):

- 1 First write to the Addr\_Port to setup the 16-bit address pointer.
- 2a For one or more Read cycles:
  - 5 i) Read the Data\_Port.
  - ii) The 8-bit data pointed to by the Addr\_Port is placed in the LSB of the frame.
  - iii) Addr\_Port <= Addr\_Port+1
- 2b For one or more Write cycles:
  - i) Write the Data\_Port.
  - ii) The 8-bit data in the LSB of the frame is written at the Addr\_Port.
  - iii) Addr\_Port <= Addr\_Port+1
- 3 Repeat 2a or 2b for more data transfers.

Mode 2 (16-bit access):

- 1 First write to the Addr\_Port to setup the 16-bit address pointer.
- 2a For one or more Read cycles:
  - i) Read the Data\_Port
  - 20 ii) The 16-bit data pointed to by the Addr\_Port is placed in the MSB & LSB of the frame.



iii) Addr\_Port <= Addr\_Port+2

2b For one or more Write cycles:

i) Write the Data\_Port

ii) The 16-bit data in the MSB and LSB of the frame is  
written at the Addr\_Port.

iii) Addr\_Port <= Addr\_Port+2

3 Repeat 2a or 2b for more data transfers.

Mode 3 (addr/data-bundled write-only access):

1 i) First write to the Addr\_Port to setup the MSB of  
the 16-bit address pointer (LSB is ignored).

2 ii) Write to Data\_Port.

iii) The MSB of the frame contains the LSB of the address  
pointer.

iv) The LSB of the frame contains the data to be written.

3 Repeat 2 for more writes.

Mode 4 (JMP or JSR and execute):

1 Write a 16-bit address to the JMP\_JSR\_Port.

2 Program execution is immediately transferred to the  
previously downloaded code beginning at the JMP\_JSR\_Port  
address.

Register RF0.R22 controls the selection of modes 1, 2 and 3 above. Following a hard reset, mode 3 is the default mode. Mode 4 is entered by directly writing a 16-bit address to the JMP\_JSR\_Port.

5       FIGURE 4 is an exemplary memory map of ROM 320 and RAM 330 according to the principles of the present invention. RAM 330A illustrates the typical contents of RAM 330 if a software patch or upgraded or enhanced software control program is not downloaded into RAM 330. RAM 330B illustrates the typical contents of RAM 330 if a software patch or upgraded or enhanced software control program is downloaded into RAM 330.

In the exemplary embodiment, ROM 320 stores program code in the address space from 0xC000 to 0xFBCF. ROM 320 stores interrupt vectors and interrupt service routines in the address space from 0xFB00 to 0xFFFF. If code is running in ROM 320, RAM 330A stores pointers, variable and user-defined code used by the embedded program in ROM 320 in the address space from 0x8000 to 0x83FF. RAM 330A may also store patch code and additional variables in the address space from 0x8400 to 0xBF7F. The address space from 0xBF80  
20 to 0xBFFF holds the stack used by the embedded program in ROM 320.

However, if an upgrade or an enhancement to the software control program is downloaded to RAM 330, the program code in

ROM 320 will run until branching to the address in RAM 330B specified by JMP\_JSR\_Port in register RF0.R31. RAM 330B stores pointers, variable and user-defined code used by the downloaded software program code in RAM 330 in the address space from 0x8000 to 0x83FF. The software program itself is stored in RAM 330B in the address space from 0x8400 to 0x9DCF. RAM 330B stores interrupt service routines in the address space from 0x9F00 to 0xA0FF. RAM 330B stores patch code and additional variables in the address space from 0xA100 to 0xBF7F. The address space from 0xBF80 to 0xBFFF holds the stack used by the downloaded software program in RAM 330B.

FIGURE 5 depicts flow diagram 500, which illustrates the operation of exemplary network interface card 140 according to one embodiment of the present invention. After a reboot or initial start up of PC 100, PC 100 executes software drivers that retrieve an upgraded or enhanced software control program, or error corrections for the software control program from NIC configuration file 150 in hard disk drive 103 (process step 505). Next, PC 100 downloads the new software control program to microcontroller 300 in the physical layer of NIC 140 using the standard MDC and MDIO signal lines. The new software control program is stored in RAM 330 (process step 510).

Microcontroller 300 then begins to run the embedded program in ROM 320. However, when PC 100 writes a 16-bit value to the JMP\_JSR\_Port RAM address in RF0.R31, the embedded program immediately jumps to the RAM address specified by JMP\_JSR\_Port. Thereafter, program control is transferred to the downloaded software control program in RAM 330 beginning at the JMP\_JSR\_Port address (process step 515). At this point, NIC 140 begins to operate under the control of the new software control program (process step 520).

As mentioned previously, manufacturers of conventional network interface cards traditionally rely on state machines to control the functions of the physical layer of the NIC. One of the overriding reasons for this is the ability of individual state machines to execute in parallel. Microcontrollers have not been used in the prior art network interface cards because of the sequential nature of microcontrollers.

In considering the incorporation of microcontroller 300 in the physical layer of NIC 140, it is equally necessary to consider a non-conventional (i.e., non-sequential) embedded control program (or downloaded software control program) to control the operations of microcontroller 300. Conventional (sequential) control programs are far too slow for monitoring and controlling the real-time

functions of NIC 140. It is therefore necessary to incorporate a multi-tasking capability that can service the functions of microcontroller 300 in a real-time manner.

5 The ability to download an updated multi-tasking software control program into RAM 330 permits correction and optimization of functions without having to resort to re-fabrication every time a change or enhancement is needed or desired. The ability to multi-task the embedded control program or the software control program using an off-the-shelf microcontroller core is key to cost-effectively controlling the real-time operation of NIC 140. Furthermore, programming a standard microcontroller using an industry standard programming language and compiler is far easier to accomplish and test as compared with designing, simulating and testing equivalent hardware state machines.

FIGURE 6 depicts flow diagram 600, which illustrates the hierarchy of multitasking routines in the control program that operates exemplary network interface card 140 according to one embodiment of the present invention. Multitasking may be performed by the original embedded control program in ROM 320 or by the new downloaded software control program in RAM 330, or both. The majority of the physical layer operations in NIC 140 may be monitored and controlled by the embedded control program, which

resides in 16K of internal ROM 320. Additionally, a downloaded software control program in internal or external RAM, or both, may run independently, or together with, the ROM-based embedded control program. PC 100 may download a software control program through management interface logic 340. A downloaded software control program is normally used for testing, debugging, data logging, bug fixing and adding enhancements.

Since microcontroller 300 is essentially a sequential processor, the present invention uses a method of multi-tasking to accommodate servicing the multiple routines illustrated in FIGURE 6. Microcontroller 300 executes main routine 605, which may may branch to any one of subroutines 610, 615, 620, 625, and 630, which are arbitrarily labeled Subroutine 1, Subroutine 2, Subroutine 3, Subroutine 4, and Subroutine 5, respectively. Main routine 605 operates as a top-level executive routine. Main routine 605 calls each of the major subroutines, including subroutines 610, 615, 620, 625, and 630, in order to give each subroutine an opportunity to complete a portion of its tasks.

At the most basic level, all subroutines in an embedded control program or in a downloaded software control program share the same sequential flow, which can generally be described as a sequence of instructions that perform some process followed by a

decision block, followed by additional processes and decisions. Without multitasking, normal program flow in any given subroutine may be suspended in a loop in a decision block for an indefinite period of time. An example of this type of event is waiting for a timer to expire or waiting for a particular signal to go true or false. In either case, other subroutines cannot be serviced during the stall period, unless some form of multitasking is implemented.

Multitasking in microcontroller 300 is done by using a series of pointers called multitasking vectors (or MTVs) which reside in RAM 330. Each one of subroutines 610, 615, 620, 625, and 630 has its own MTV and at Power ON, the control program initializes the MTV of each subroutine to point to the beginning of that subroutine. As each subroutine executes, it updates its own MTV. When the subroutine comes to a decision point that cannot be immediately satisfied (such as waiting for a timer to expire), the subroutine leaves its MTV pointing to the beginning of the decision loop and returns to the calling routine (e.g., main routine 605). The next subroutine in sequence is then serviced. The next time the original subroutine is serviced, it begins processing at the last MTV address position.

FIGURES 7-9 illustrate multitasking program flow in three exemplary subroutines, namely subroutines 610, 615, and 620 (i.e.,

Subroutines 1, 2 and 3), according to one embodiment of the present invention. Subroutine 1 comprises a sequence of processes (Processes 1-5) separated by decision blocks (Decisions 1-4). Similarly, Subroutine 2 and Subroutine 3 also comprise a sequence of processes (Processes 1-5) separated by decision blocks (Decisions 1-4).

Microcontroller 300 begins processing Subroutine 1 by first completing Process 1. When Process 1 is complete, Subroutine 1 moves its Multi-Tasking Vector, MTV1, to the beginning of Decision 1. Assuming that Decision 1 cannot be immediately completed, Subroutine 1 then leaves MTV1 pointing to the beginning of Decision 1, and the control program begins processing Subroutine 2 starting from Process 1.

Process 1 in Subroutine 2 completes and the control program moves MTV2 to the beginning of Decision 1. Again, for purposes of example, Decision 1 cannot be completed immediately, so Subroutine 2 leaves MTV2 pointing to the beginning of Decision 1. The control program then begins processing Subroutine 3. Subroutine 3 executes Process 1 and encounters Decision 1, which cannot be immediately satisfied. This process continues until program control eventually comes back to Subroutines 1, 2 and 3.

Moving forward in time, FIGURE 8 shows the positions of MTV1,



MTV2 and MTV3 in their respective subroutines. As can be seen, Subroutine 1 has progressed to Decision 2, while Subroutine 2 is still waiting for Decision 1 to complete. Subroutine 3 has progressed to Decision 3. Again moving forward in time, FIGURE 9 shows the positions of MTV1, MTV2 and MTV3 in their respective subroutines. Subroutine 1 has progressed to Decision 3, Subroutine 2 has progressed to Decision 2, and Subroutine 3 has progressed to Decision 4. Advantageously, microcontroller 300 spends very little time in each of Subroutines 1, 2 and 3 testing for a decision. Rather, microcontroller 300 rapidly moves from subroutine to subroutine, all the while maintaining the real-time integrity of each subroutine.

An important subset of multitasking subroutines are patch routines. According to an advantageous embodiment of the present invention, patch routines are the preferred way to download a customized software control program to modify the programmatic flow of the ROM-based embedded software control program. As the name implies, a patch routine allows the user to patch together or patch around portions of an existing embedded ROM control program. Patch routines can also stand alone as separate callable subroutines not related to any existing embedded control program function.

In an exemplary embodiment of the present invention, patch

5 routines may be implemented using MTV patch flags. From either Power ON or a manual reset, microcontroller 300 clears to all zeroes 32 MTV patch flags, which are organized in four contiguous bytes in internal RAM 330. The MTV patch flags are the turn ON or Turn OFF mechanism for their related patch routines. That is, the user may download from 1 to 32 patch routines into internal or external RAM (e.g., RAM 330) and by setting or resetting any of the 32 MTV patch flags, activate or deactivate any of the related patch routines.

Provisions are made for each of the 32 patch routines to have a multitasking vector (MTV) exactly like a normal embedded control program subroutine. Each patch routine may or may not be called from main routine 605, depending on the state of its MTV patch flag. Patch routines are constructed exactly like any other multitasking control program routine, except that all patch routines are actually callable subroutines that have an RTS (Return From Subroutine) as the last instruction. In an exemplary embodiment of the present invention, patch routines may be created and assembled in 68HC11 assembly language and then downloaded into RAM 330 via management interface logic 340.

Although the present invention has been described in detail, those skilled in the art should understand that they can make



Figure 1 illustrates the steps of the proposed algorithm for finding the minimum spanning tree of a graph. The graph has 10 nodes and 15 edges. The steps are as follows:

- (a) Initial graph with 10 nodes and 15 edges.
- (b) Selection of the first edge.
- (c) Selection of the second edge.
- (d) Selection of the third edge.
- (e) Selection of the fourth edge.
- (f) Selection of the fifth edge.
- (g) Selection of the sixth edge.
- (h) Selection of the seventh edge.
- (i) Selection of the eighth edge.
- (j) Selection of the ninth edge.
- (k) Selection of the tenth edge.
- (l) Final minimum spanning tree.